People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods

Richard Turner
The George Washington University

Barry Boehm
University of Southern California

While methodologies, management techniques, and technical approaches are valuable, a study of agile and plan-driven approaches has confirmed that the most critical success factors are much more likely to be in the realm of people factors. This paper discusses five areas where people issues can have a significant impact: staffing, culture, values, communications, and expectations management.

Recently we have been studying the characteristics of agile and plan-driven methods to provide guidance in balancing the agility and discipline required for successful software acquisitions or developments [1]. One of the most significant results of our analysis was the realization that, while methodologies, management techniques, and technical approaches are valuable, the most critical success factors are much more likely to be in the realm of people factors.

We believe that the agilists have it right in valuing individuals and interactions over processes and tools [2]. However, they are not the first to emphasize this. There is a long list of wake-up calls: Weinberg's 1971 "Psychology of Computer Programming" [3], the Scandinavian participatory design movement [4], DeMarco and Lister's 1987 "Peopleware" [5], and Curtis' studies of people factors [6] and development of the People Capability Maturity Model® [7].

There is also a wealth of corroborative evidence that people factors dominate other software cost and quality drivers. These include the 1986 Grant-Sackman experiments showing 26:1 the variations in people's performance [8], and the 1981 and 2000 Constructive Cost Model (COCOMO) and COCOMO II cost model calibrations showing 10:1 the effects of personnel capability, experience, and continuity [9, 10]. However, the agilists may finally provide a critical mass of voices amplifying this message.

In this article, we discuss five areas where we believe significant progress can be made: staffing, culture, values, communications, and expectations management.

Staffing

In essence, software engineering is done "of the people, by the people, and for the people."

- Of the People. People organize themselves into teams to develop mutually satisfactory software systems.

- By the People. People identify what software capabilities they need, and other people develop these for them.
- For the People. People pay the bills for software development and use the resulting products.

The two primary categories of players in the software development world are customers and developers.

Customers

Unfortunately, software engineering is still struggling with a *separation-of-concerns* legacy that contends translating customer requirements into code is so hard that it must be accomplished in isolation from people concerns – even the customer's. A few quotes will illustrate the situation:

- The notion of *user* cannot be precisely defined, and therefore it has no place in computer science or software engineering [11].
- Analysis and allocation of the system requirements is not the responsibility of the software engineering group, but it is a prerequisite for their work [12].
- Software engineering is not project management [13].

In today's and tomorrow's world, where software decisions increasingly drive system outcomes, this separation of concerns is increasingly harmful. Customers must be more closely related to the development process. One of the major differences between agile and plan-driven methods is that agile methods strongly emphasize having dedicated and collocated customer representatives, while plan-driven methods count on a good deal of up-front, customer-developer work on contractual plans and specifications.

For agile methods, the greatest risk is that insistence on a dedicated, collocated customer representative will cause the customer organization to supply the person that is most expendable. This risk establishes the need for criteria to determine the adequacy of customer representatives.

In our critical success factor analysis of more than 100 e-services projects at the University of Southern California, we have found that success depends on having customer representatives who are collaborative, representative, authorized, committed, and knowledgeable (CRACK) performers. If the customer representatives are not collaborative, they will sow discord and frustration, resulting in the loss of team morale. If they are not representative, they will lead the developers to deliver unacceptable products. If they are not authorized, they will incur delays seeking authorization or, even worse, lead the project astray by making unauthorized commitments. If they are not committed, they will not do the necessary homework and will not be there when the developers need them most. Finally, if they are not knowledgeable, they will cause delays, unacceptable products, or both.

This summary of customer impact on the landmark Chrysler Comprehensive Compensation project, considered to be the first eXtreme Programming (XP) project, is a good example of the need for CRACK customer representatives.

The on-site customer in this project had a vision of the perfect system she wanted to develop. She was able to provide user stories that were easy to estimate. Moreover, she was with the development team every day, answering any business questions the developer had.

Halfway [through] the project, several things changed, which eventually led to the project being cancelled. One of the changes was the replacement of the on-site customer, showing that the actual way in which the customer is involved is one of the key success factors in an XP project. The new on-site customer was present most of the time, just like the previous on-site customer, and available to the development team for questions. Unfortunately, the requirements

and user stories were not as crisp as they were before. [14]

Plan-driven methods also need CRACK customer representatives and benefit from full-time, on-site participation. Good planning artifacts, however, enable them to settle for part-time CRACK representatives who provide further benefits by keeping active in customer operations. The greatest customer challenge for plan-driven methods is to keep project control from falling into the hands of overly bureaucratic contract managers who prioritize contract compliance above getting project results.

A classic example of customer bureaucracy is provided in Robert Britcher's book, "The Limits of Software" [15], describing his experience on perhaps the world's biggest failed software project: the FAA/IBM Advanced Automation System for U.S. National Air Traffic Control. Due to many bureaucratic and other problems, including responding to change over following a plan, the project was overrun by several years and billions of dollars.

For example, one of the software development groups came up with a way to reduce the project's commitment to a heavyweight brand of software inspections that were slowing the project down by consuming too much staff effort in paperwork and redundant tasks. The group devised a lightweight version of the inspection process. It was comparably successful in finding defects, but with much less time and effort. Was the group rewarded for doing this? No. The contracting bureaucracy sent them a cease-and-desist letter faulting them for contract noncompliance and ordered them to go back to the heavyweight inspections. Agilists justifiably deride this kind of plan-driven bureaucracy.

Developers

Critical people-factors for developers using agile methods include amicability, talent, skill, and communication [16]. An independent assessment identifies this as a potential problem for agile methods: "There are only so many Kent Becks in the world to lead the team. All of the agile methods put a premium on having premium people …" [17]. Figure 1 distinguishes the most effective operating points of agile and plan-driven projects [18, 19]. Both operate best with a mix of developer skills and understanding, but agile methods tend to need a richer mix of higher-skilled people.

When you have such people available on your project, statements like, "A few

designers sitting together can produce a better design than each could produce alone," are valid. If not, you are more likely to get design by committee, with the opposite effect. The plan-driven methods do better with great people, but are generally more able to plan the project and architect the software so that less-capable people can contribute with low risk. A significant consideration here is the unavoidable statistic that 49.999 percent of the world's software developers are below average (slightly more precisely, below median).

It is important to be able to classify the type of personnel required for success in the various methods. Alistair Cockburn has addressed levels of skill and understanding required for performing various method-related functions, such as using, tailoring, adapting, or revising a method. Drawing on the three levels of understanding in the martial art Aikido, he has identified three levels of software method understanding that help sort out what various levels of people can be expected to do within a given method framework [18]. Modifying his work to meet our needs, we split his Level 1 to address some distinctions between agile and plan-driven methods, and added an additional level to address the problem of method-disrupters. Our version is provided in Table 1.

The characteristics of Level -1 people should be rapidly identified and reassigned to work other than performing on either agile or plan-driven teams; we recommend such activities as commercial off-the-shelf assessment or *my-four-year-old-can't-break-it* testing.

Level 1B people are average and below, less-experienced, hard-working developers. They can function well in performing straightforward software development in a stable situation. However, they are likely to slow an agile team trying to cope with rapid change, particularly if they form a majority of the team. They can form a well-performing majority of a stable, well-
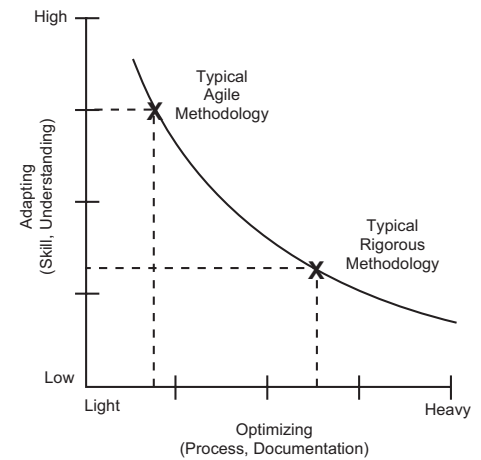

Figure 1: *Balancing, Optimizing, and Adapting Dimensions* [16]

structured, plan-driven team.

Level 1A people can function well on agile or plan-driven teams if there are enough Level 2 people to guide them. When agilists refer to being able to succeed on agile teams with a ratio of five Level 1 people per each Level 2 person, they are generally referring to Level 1A people.

Level 2 people can function well in managing a small, precedented agile or plan-driven project but need the guidance of Level 3 people on a large or unprecedented project. Some Level 2s have the capability to become Level 3s with experience. Some do not.

Staffing and the Home Grounds

We found that these skill levels were one of the five key discriminators in determining whether a new project would best fit the *home grounds* of agile and plan-driven methods. Home grounds are the set of conditions under which the methods are most likely to succeed. In Figure 2 (see page 6), we graphically portray these home grounds based on five critical factors. In general, the closer to the center, the more the factors favor agility.

The *personnel* axis in Figure 2 shows that the home ground for agile methods requires at least 30 percent to 35 percent

Table 1: *Levels of Software Method Understanding and Use* [17]

| Level	Characteristics
3	Is able to revise a method (break its rules) to fit an unprecedented new situation.
2	Is able to tailor a method to fit a precedented new situation.
1A	With training, is able to perform discretionary method steps (e.g., sizing stories to fit increments, composing patterns, compound refactoring, and complex COTS integration). Can become Level 2 with experience.
1B	With training, is able to perform procedural method steps (e.g., coding a simple method, simple refactoring, following coding standards and capability model procedures, and running tests). Can master some Level 1A skills with experience.
-1	May have technical skills, but is unable or unwilling to collaborate or follow shared methods.
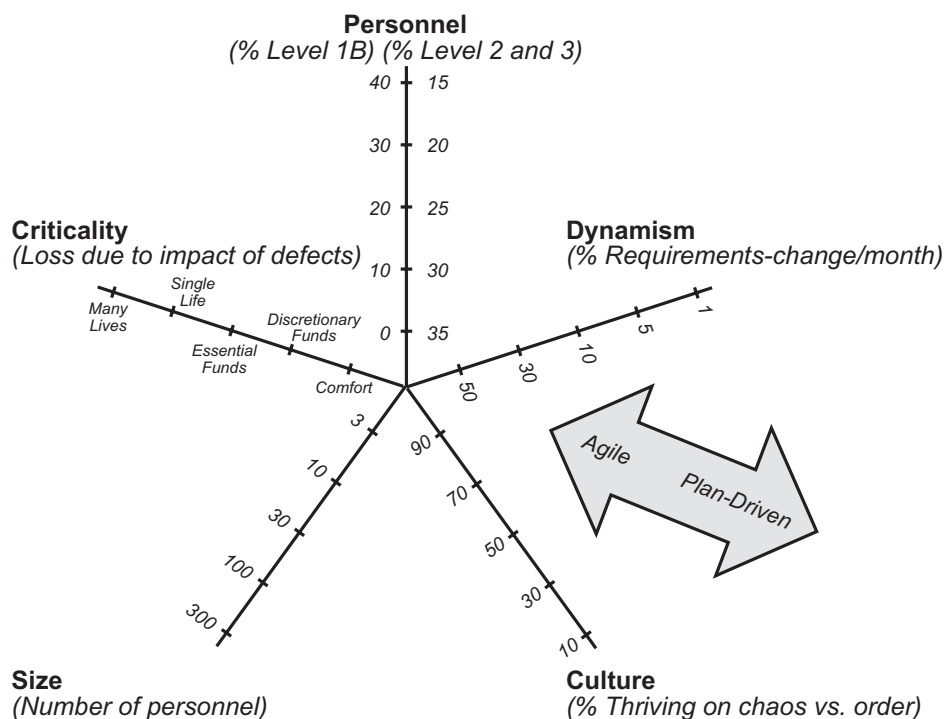
Figure 2: *Key Discriminators of Agile and Plan-Driven Home Grounds*

of the project's people to have Level 2 and 3 skills, with no more than 10 percent of the people with Level 1B skills. The home ground for plan-driven methods can succeed with up to 30 percent to 40 percent Level 1B people, and as few as 15 percent to 20 percent Level 2 and 3 people.

In fact, three of the five key discriminators in Figure 2 are people-related: *personnel* (as discussed above), *size*, and *culture*. The size of the project is measured in the number of people. Agile methods succeed best on projects of 10 people or less, while plan-driven methods work better on projects of 100 people and up. In his landmark XP book, Kent Beck says,

> Size clearly matters. You probably couldn't run an XP project with a hundred programmers. Not fifty. Nor twenty, probably. Ten is definitely doable. [20]

Projects in the middle range of the key discriminator factors need a hybrid mix of agile and plan-driven methods [1]. We will next look more closely at culture.

Culture

The second area of people possibilities, and the third people-related key discriminator between agile and plan-driven home grounds, is culture. In an agile culture, the people feel comfortable and are empow-

ered when there are *many degrees of freedom* available for them to define and work problems. This is the classic craftsman environment, where each person is expected and trusted to do whatever work is necessary for the success of the project. This includes looking for common or unnoticed tasks and completing them.

In a plan-driven culture, the people feel comfortable and empowered when there are *clear policies and procedures* that define their role in the enterprise. This is more of a production-line environment where each person's tasks are well defined. The expectation is that they will accomplish the tasks to specification so that their work products will easily integrate into others' work products with limited knowledge of what others are actually doing.

These cultures are reinforced as people tend to self-select for their preferred culture, and as people within the culture are promoted to higher management levels. Once a culture is well established, it is difficult and time consuming to change. This cultural inertia may be the most significant challenge to the integration of agile and plan-driven approaches.

To date, agile cultural change has had a bottom-up, revolutionary flavor. Failing projects with no hope of success have been the usual pilots, supported by an *it can't hurt* attitude from management and a *no challenge is too hard* adrenaline-charged response from practitioners. Successes have been extraordinary in many cases and have been used to defend migration to less troubled projects.

Early Capability Maturity Model® for Software (SW-CMM®) [21] adopters faced similar challenges, although there was early involvement of middle management. The concept of culture change evolved rapidly and is now well understood by the managers and software engineering process groups. These have been the main change agents in evolving their organizations from following improvised, ad hoc processes toward following plan-driven, SW-CMM-compliant processes.

The new CMM Integration℠ (CMMI®) [22] upgrades the SW-CMM in more agile directions, with new process areas for integrated teaming, risk management, and overall integrated systems and software engineering. A number of organizations are welcoming this opportunity to add more agility to their organizational culture. Others that retain a more bureaucratic interpretation of the SW-CMM are facing the challenge of *change-averse change agents* who have become quite comfortable in their bureaucratic culture.

Values

Along with people come values – different values. One of the most significant and underemphasized challenges in software engineering is to reconcile different users', customers', developers', and other success-critical stakeholders' value propositions about a proposed software system into a mutually satisfactory win-win system definition and outcome. Unfortunately, software engineering is caught in a value-neutral time warp, where every requirement, use case, object, test case, and defect is considered to be equally important.

Most process improvement initiatives and debates, including the silver-bullet debate are inwardly focused on improving software productivity rather than outwardly focused on delivering higher value per unit cost to stakeholders. Again, agile methods and their attention to prioritizing requirements and responding to changes in stakeholder value propositions are pushing us in more high-payoff directions.

Other aspects of value-based software engineering practices and payoffs are described in "Value-Based Software Engineering" [23]. These include the DMR Consulting Group's Benefits Realization Approach and Results Chains [24], stakeholder win-win requirements negotiation [25], business case analysis [26], and the Kaplan-Norton Balanced Scorecard technique [27].

Communications

Even with closely knit in-house develop-

℠ CMM Integration is a service mark of Carnegie Mellon University.
® CMM is registered in the U.S. Patent and Trademark Office.

ment organizations, the "I can't express exactly what I need, but *I'll know it when I see it* " syndrome limits people's ability to communicate an advance set of requirements for a software system. If software definition and development occurs across organizational boundaries, even more communications work is needed to define and evolve a shared system vision and development strategy. The increasingly rapid pace of change exacerbates the problem and raises the stakes of inadequate communication.

Agile methods rely heavily on communication through *tacit, interpersonal knowledge* for their success. They cultivate the development and use of tacit knowledge, depending on the understanding and experience of the people doing the work and their willingness to share it. Knowledge is specifically gathered through team planning and project reviews (an activity agilists refer to as *retrospection*). It is shared across the organization as experienced people work on more tasks with different people.

Agile methods generally exhibit more frequent, person-to-person communication. As stated in the Agile Manifesto, emphasis is given to *individuals and interactions*. Few of the agile communications channels are one-way, showing a preference for collaboration. Stand-up meetings, pair programming, and the planning game are all examples of the agile communication style and its investments in developing shared tacit knowledge.

Relying completely on tacit knowledge is like performing without a safety net. While things go well, you avoid the extra baggage and setup effort, but there may be situations that will make you wish for that net. Assuming that everyone's tacit knowledge is consistent across a large team is risky, and as people start rotating off the team, the risk gets higher.

At some point, a group's ability to function exclusively on tacit knowledge will run up against well-known scalability laws for group communication. For a team with N members, there are $N(N-1)/2$ different interpersonal communication paths to keep current. Even broadcast techniques such as stand-up group meetings and hierarchical team-of-teams techniques run into serious scalability problems.

Plan-driven methods rely heavily on *explicit documented knowledge*. With plan-driven methods, communication tends to be one-way. Communication is generally from one entity to another rather than between two entities. Process descriptions, progress reports, and the like are nearly always communicated as unidirectional flow.

We should note that this distinction between *agile-tacit* and *plan-driven-explicit* is not absolute. Agile methods' source code and test cases certainly qualify as explicit documented knowledge, and even the most rigorous plan-driven method does not try to get along without some interpersonal communication to ensure a consistent, shared understanding of documentation intent and semantics.

When agile methods employ documentation, they emphasize doing the minimum essential amount. Unfortunately, most plan-driven methods suffer from a *tailoring-down* syndrome, which is sadly reinforced by most government procurement regulations. These plan-driven methods are developed by experts who want them to provide users with guidance for most or all foreseeable situations. The experts, therefore, make them very comprehensive, but tailored down for less critical or less complex situations. The experts understand tailoring the methods and often provide guidelines and examples for others to use.

Unfortunately, less expert and less self-confident developers, customers, and managers tend to see the full-up set of plans, specifications, and standards as a security blanket. At this point, a sort of Gresham's Law (*bad money drives out good money*) takes over, and the least-expert participant can drive the project by requiring the full set of documents rather than an appropriate subset. While the nonexperts rarely read the ever-growing stack of documents, they will maintain a false sense of security in the knowledge they have followed best practices to ensure project predictability and control. Needless to say, the expert methodologists are then frustrated with how their tailorable methods are used – and usually verbally abused – by developers and acquirers alike. Agilists have certainly highlighted this significant problem in plan-driven methods.

Except for the landmark people-oriented sources mentioned above, there are frustratingly few sources of guidance and insight on what kinds of communications work best in what situations. Cockburn's "Agile Software Development" [18] is a particularly valuable recent source. It gets its priorities right by not discussing methods until the fourth chapter, and spending the first hundred or so pages discussing why we have problems communicating, and what can be done about it. It nicely characterizes software development as a cooperative game of invention and communication, and provides numerous helpful communication concepts and techniques. Some examples are his definition of the three skill levels based on Aikido

discussed earlier, human success and failure modes, information radiators and convection currents, and the effects of distance on communication effectiveness.

Expectations Management
Our primary conclusion in analyzing software project critical success factors has been that the differences between successful and troubled software projects are most often the difference between good and bad expectations management. This coincides with a major finding in a recent root-cause analysis of trouble factors in Department of Defense software projects [28].

Most software people do not do well at expectations management. They have a strong desire to please and to avoid confrontation, and have little confidence in their ability to predict software project schedules and budgets, making them a pushover for aggressive customers and managers trying to get more software for less time and money.

The most significant factor in successful agile or plan-driven teams is that they have enough process mastery, preparation, and courage to be able to get their customers to agree to reduce functionality or increase schedule in return for accommodating a new high-priority change. They are aware that setting up unrealistic expectations is not a win for the customers either, and are able to convince the customers to scale back their expectations. Both agile short iterations and plan-driven productivity calibration are keys to successfully managing software expectations.

Conclusion
Giving top-priority attention to such people-related factors as staffing, culture, values, communications, and expectations management is critical to successful software development and management. Beyond this top-level summary of key factors, there are many valuable sources of guidance on how to succeed with the people-related aspects of your software projects.

Besides the classic Weinberg, Ehn, and DeMarco-Lister books previously cited, there are some further references that can help you improve your people factors whether you use agile, plan-driven, or hybrid development approaches. Good agilist treatments of people and their ecosystems are provided in Jim Highsmith's "Agile Software Development Ecosystems" [19] and Alistair Cockburn's "Agile Software Development" [18]. Complementary plan-driven approaches are provided in Watts

Humphrey's "Managing Technical People" [29] and his Personal Software Process℠ [30], as well as the People CMM developed by Bill Curtis, Bill Hefley, and Sally Miller [31].

As engineers, our selection of reading materials tends to gravitate toward programming, architecture, or processes for our next learning experience. We strongly recommend you choose one of the books above as a way to balance your technical and people skills.◆

References

1. Boehm, B., and R. Turner. <u>Balancing Agility and Discipline: A Guide for the Perplexed</u>. Boston: Addison-Wesley, 2004.
2. Beck, Kent, et al. "The Agile Manifesto." The Agile Alliance, 2001 <www.agilealliance.com>.
3. Weinberg, G. <u>The Psychology of Computer Programming</u>. New York: Van Nostrand-Reinhold, 1971.
4. Ehn, P., Ed. <u>Work-Oriented Design of Computer Artifacts</u>. Mahwah, NJ: Lawrence Earlbaum Associates, Mar. 1990.
5. DeMarco, T., and T. Lister. <u>Peopleware: Productive Projects and Teams</u>. New York: Dorset House, 1999.
6. Curtis, B., H. Krasner, and N. Iscoe. "A Field Study of the Software Design Process for Large Systems." <u>Comm. ACM</u> 31. 11 (Nov. 1988): 1268-1287.
7. Curtis, B. et al. <u>People Capability Maturity Model</u>. Reading, MA: Addison-Wesley, 2001.
8. Grant, E., and H. Sackman. "An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions." Report SP-2581, System Development Corp., Sept. 1966.
9. Boehm, B. <u>Software Engineering Economics</u>. Upper Saddle River, NJ: Prentice Hall, 1981.
10. Boehm, B., et al. <u>Software Cost Estimation With COCOMO II</u>. Upper Saddle River, NJ: Prentice Hall, 2000.
11. Dijkstra, E. Panel Discussion. Fourth International Conference on Software Engineering, 1979.
12. Paulk, M., et al. <u>The Capability Maturity Model for Software: Guidelines for Improving the Software Process</u>. Reading, MA: Addison-Wesley, 1994
13. Tucker, A. "On the Balance Between Theory and Practice." <u>IEEE Software</u> Sept.-Oct. 2002.
14. van Duersen, A. "Customer Involvement in Extreme Programming." <u>ACM Software Engineering Notes</u> Nov. 2001: 70-73.
15. Britcher, R. N. <u>The Limits of Software</u>. Reading, MA: Addison-Wesley, 1999.
16. Highsmith, J., and A. Cockburn. "Agile Software Development: The Business of Innovation." <u>Computer</u> Sept. 2001: 120-122
17. Constantine, L. "Methodological Agility." <u>Software Development</u> June 2001: 67-69.
18. Cockburn, Alistair. <u>Agile Software Development</u>. Boston: Addison-Wesley, 2002.
19. Highsmith, Jim. <u>Agile Software Development Ecosystems</u>. Boston: Addison-Wesley, 2002.
20. Beck, Kent. <u>Extreme Programming Explained</u>. Boston: Addison-Wesley, 1999: 157.
21. Paulk, et al. <u>The Capability Maturity Model</u>. Reading, MA: Addison-Wesley, 1994.
22. Ahern, D. M., A. Clouse, and R. Turner. <u>CMMI Distilled: A Practical Introduction to Integrated Process Improvement</u>. 2nd ed. Boston: Addison-Wesley, 2003.
23. Boehm, B. "Value-Based Software Engineering." <u>ACM Software Engineering Notes</u> Mar. 2003.
24. Thorp, J. <u>The Information Paradox</u>. McGraw-Hill, 1998.
25. Boehm, B., P. Bose, E. Horowitz, and M. J. Lee. <u>Software Requirements As Negotiated Win Conditions</u>. Proc. of the First International Conference on Requirements Engineering, Colorado Springs, CO. IEEE Computer Society Press, Apr. 1994.
26. Reifer, D. <u>Making the Software Business Case</u>. Boston: Addison-Wesley, 2002.
27. Kaplan, R., and D. Norton. <u>The Balanced Scorecard: Translating Strategy into Action</u>. Boston: Harvard Business School Press, 1996.
28. McGarry, J., and Charette, R. "Systemic Analysis of Assessment Results from DoD Software-Intensive System Acquisitions." Tri-Service Assessment Initiative Report, Office of the Under Secretary of Defense (Acquisition, Technology, Logistics), 2003.
29. Humphrey, W. <u>Managing Technical People</u>. Boston: Addison-Wesley, 1997.
30. Humphrey, W. <u>Introduction to the Personal Software Process</u>. Boston: Addison-Wesley, 1997.
31. Curtis, B., B. Hefley, and S. Miller. <u>The People Capability Maturity Model</u>. Boston: Addison-Wesley, 2001.

℠ Personal Software Process is a service mark of Carnegie Mellon University.

About the Authors

Richard Turner, D. Sc., is a member of the Engineering Management and Systems Engineering Faculty at The George Washington University in Washington, D.C. Currently, he is the assistant deputy director for Software Engineering and Acquisition in the Software Intensive Systems Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics). Turner is co-author of the book "CMMI Distilled."

1931 Jefferson Davis Highway
Suite 104
Arlington, VA 22202
Phone: (703) 602-0581 ext. 124
E-mail: rich.turner.ctr@osd.mil

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 900989-0781
Phone: (213) 740-8163,
**   (213) 740-5703**
Fax: (213) 740-4927
E-mail: boehm@sunset.usc.edu